Available on CMS information server **CMS IN 2005/XXX**

CMS Internal Note

The content of this note is intended for CMS internal use and distribution only

January 5, 2005

Review of Jet Software at CMS

Draft Version 2

Robert M. Harris

Fermilab

Marek Zielinski

University of Rochester

Abstract

We present a review of the software for reconstructing and accessing jets at CMS. Central to the review is a comparison of the current structure and functionality of the jet software at CMS, CDF and D0. We conclude that changes to the CMS jet software are desirable to enhance the usability and efficiency of the CMS code. We make suggestions for incorporating the best features of the CDF and D0 code into the CMS jet software.

Contents

1 Introduction

We have conducted a review of the CMS jet software [1]. Our goal was to understand the jet code to insure it meets the requirements for jet reconstruction, access, and analysis in a hadron collider environment. This note reports the result of both our studies of the CMS, CDF and D0 jet code, and the conclusions from our discussions with the jet code experts at CMS [2], CDF [3] and D0 [4]. We write the note from the perspective of physicists who have used jet software for the last two decades at the Tevatron. The review therefore concentrates on physics requirements and usage and not on software methodology.

To focus our review we concentrate on a vertical slice of the jet code that goes from the jets back to the inputs that compose the jet. We will consider more than one type of input to create jets, since it is important that the jet software be able to support multiple inputs. However, we will concentrate on the tower level input, since that is the most frequently performed jet reconstruction at CDF and D0, and we anticipate that it will likely be the most common jet reconstruction at CMS as well.

2 Jet Access

The jet code at CMS, CDF and D0 are all very similar at the level of the EDM for retrieving jet objects. The three experiments employ jet collections, and have similar methods for finding those collections, and iterating over the jet objects they contain. CMS employs a RecQuery to find a RecCollection of RecJets. CDF directly uses a find method on the CDFJetColl of CdfJets. D0 uses a find method to get a JetChunk of Jets. The names are different, but the concepts are very similar.

3 Jet Code

Our review of jet creation and storage is focused on the following four issues:

1. List of jet quantities saved/retrieved with the jet object.
2. Ability to retrieve the input objects that make the jet.
3. Handling of the event vertex information in jet creation and calculation.
4. Handling of multiple sources of jet input objects.

After the jet and pre-clustering algorithms themselves, these issues are the most important elements in the jet software. They greatly affect the users ability to obtain essential jet data, calculated with the necessary inputs, and with the actual event vertex taken into account.

A simplified structure of the jet code at CMS, CDF and D0 is pictured in Figure 1. In the sections that follow we will analyze the jet code using Figure 1 as our guide.

3.1 Jet Storage and Data Access

At CMS the jet class (RecJets) contains the Lorentz vector (four-vector) and methods to access it or it's equivalents: E, P_x, P_y, P_z or E_T, m, η and ϕ. The RecJets class also contains the Jet Constituents, the elements that make up the jet (towers or MC particles or whatever), implemented as a vector of VJetableObjects. The jet algorithm takes as its input all the VJetableObjects, and the RecJet saves on the DST the Jet Constituents (the set of VJetableObjects associated with the final jets).

At CDF and D0 the jet class (CDFJet at CDF and Jet at D0) saves all the relevant jet quantities for analysis, not just the Lorentz vectors. Quantities like the electromagnetic fraction of jet energy and the number of tracks pointing at the jet. Access to a pre-calculated list of jet quantities, stored with the jet object itself, greatly facilitates jet analysis at CDF and D0. Early versions of the CDF code that did not calculate these quantities and save them with the jet, or went back to the jet inputs to calculate or recalculate these quantities, were subsequently replaced by code that actually saved these quantities with the jet in order to satisfy user demands. The benefit to the user of having these quantities available, and unambiguously associated with the jet and its pre-calculated Lorentz vector, were found to be significantly more important than the space they took up in memory or on disk [3]. We strongly recommend that the most frequently used jet quantities, such as those listed in Appendix A, be available through the persistent CMS jet object.

At CDF and D0 the jet class provides access to the input objects that formed the jet via pointers. At CDF the pointers are a list of indices that specify the eta-phi location of the PhysicsTowers that make up the jet, and these indices have been designed so they also uniquely specify the calorimeter objects (1 EM Tower and 1 Hadronic Tower) that were used to make the PhysicsTower. Thus at CDF the jet class links back through the entire chain of objects that create it, and any stage is available to the user as long as the input objects are saved on disk. At D0 the pointers go back only to the lowest level of underlying calorimeter objects (the multiple EM and Hadronic layers that make up the CalTowers), allowing the user to examine the most fundamental calorimeter aspects of the jet, but not the actual CalTowers that make up the jet. The D0 jet code maintainer says that it would have been good to provide links back to the CalTowers themselves, and there have been requests at D0 for this feature [4]. In both CDF and D0 what is saved are links back to the towers, not the actual towers themselves, and methods are provided to retrieve the towers from these links. This avoids the unnecessary duplication of space when the towers themselves are also saved, for re-reconstruction, but provides an unambiguous list of the actual towers that composed the jet. With a list of pointers there is no need to rerun the algorithm to find the towers and face the resulting risk of getting a different list of towers (due to potential changes in calibration, parameters, or other possibly poorly controlled features in the experiment). The CMS solution of storing the VJetableObjects themselves meets all the user requirements for access to input towers at the cost of a significant increase in the size of the jet object. We recommend, that if event size is an issue, the Jet Constituents be replaced with a list of pointers to the jet input objects (VJetableObjects or whatever replaces it).

3.2 Jet Algorithms

At CMS the following jet algorithms are employed

- Cone Algorithms
    1. Simple Cone Algorithm: cone around a seed.
    2. Iterative Cone Algorithm: cone around a centroid, iterated from a seed.
    3. CMS Midpoint: cone around a centroid, iterated from seeds and the midpoints of pairs of seeds.
- KT algorithms: three different algorithms.

None of the cone algorithms employs jet splitting or merging. That means that when two jet cones overlap there is no intelligence applied to assigning the energy to one cone or another (splitting) or to deciding that the two energy depositions should be designated a single jet (merging). This is a recognized drawback of the CMS cone algorithms, and steps have been taken to try to implement the CDF/D0 midpoint algorithm, which does employ splitting and merging, but CMS has not been successful so far in implementing an algorithm that does not crash.

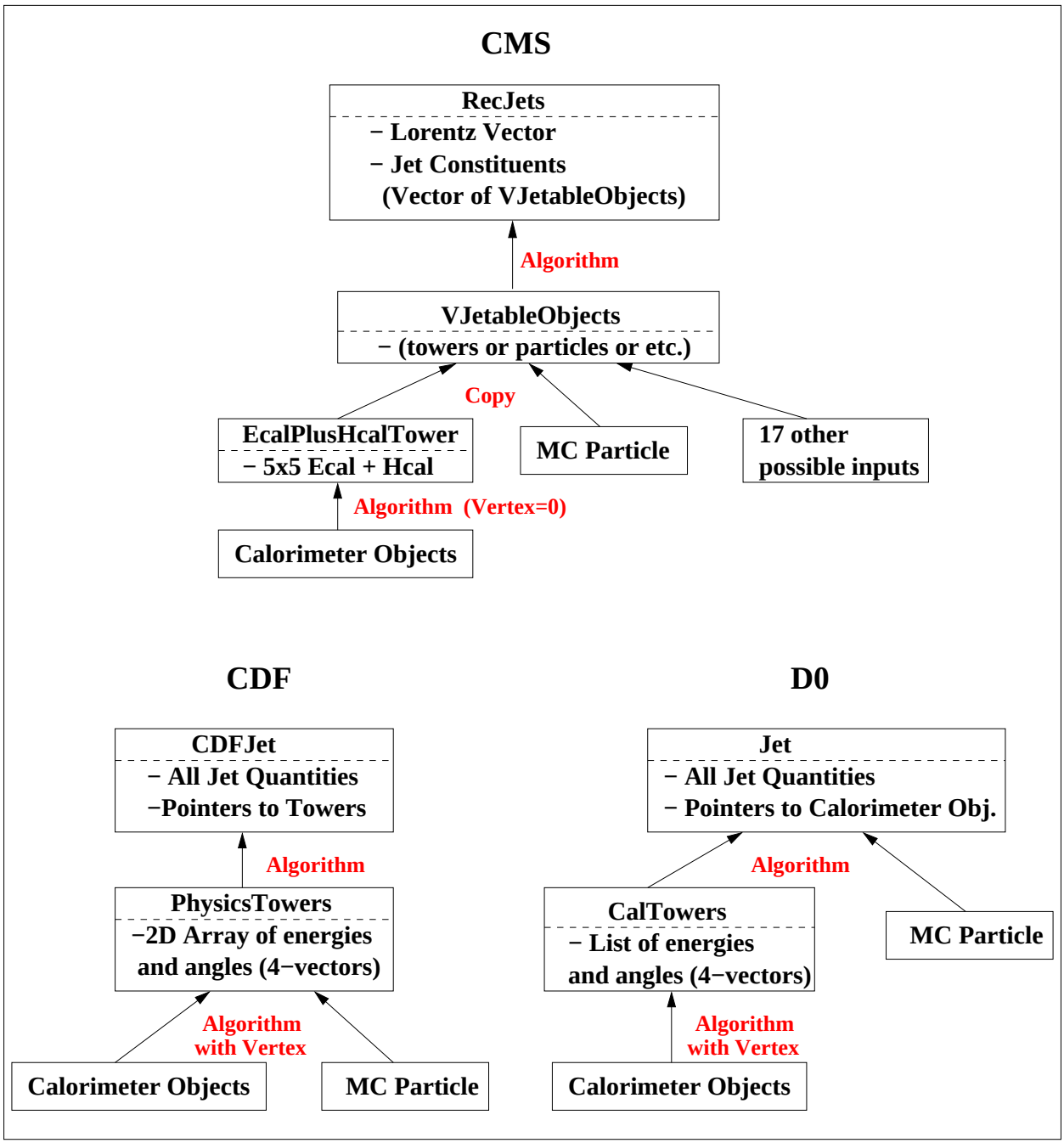

Figure 1: Vertical Slice of the Jet Software at CMS, CDF and D0. The objects are pictured by boxes with bulleted lists of what they contain. The order of reconstruction is pictured by arrows, which are algorithms, connecting the objects.

The iterative cone algorithm, with a cone size $R = 0.5$, has emerged as the defacto standard at CMS, despite it's lack of splitting and merging.

CDF and D0 also employ cone algorithms, both legacy algorithms from Run 1 and the Midpoint algorithm in run 2, and all these algorithms employ jet splitting and merging. This was because it was recognized early on that splitting was essential for top physics (for high jet finding efficiency from W decays and in the presence of QCD radiation) and that jet merging was useful for dijet physics (to reduce the effect of final state radiation without using a larger cone). Both experiments also employ Kt algorithms primarily for QCD studies. In addition both experiments make available seedless cone algorithms which are not used much because they require a lot of CPU time to run.

We strongly recommend that CMS continue in its efforts to implement the CDF/D0 midpoint algorithm, in order to take advantage of jet splitting and merging, and also to have the cleanest possible comparison with results from the Tevatron. Further, we believe that the CDF/D0 midpoint algorithm should be adopted as the CMS standard jet algorithm

3.3 Jet Input

At CMS the input objects to jet reconstruction, the VJetableObjects, are simply copies of the desired input objects. The most popular calorimeter level input object is the ECalPlusHcalTower which consists of an Hcal tower and the multiple Ecal towers in front of it (in the barrel it consists of 25 Ecal towers: 5 in η x 5 in ϕ). The algorithm that creates the EcalPlusHcalTowers simply sums the energies and assigns an angle to the tower assuming an event vertex located at the center of the CMS detector (nominal collision point). Another popular input is MC particles, and there are seventeen other possible inputs, including the hit level quantities at the calorimeter themselves without forming towers. VJetableObjects acts as an interface class for handling all these multiple inputs for making jets, and, unfortunately, assumes that all the input objects originate from the center of the CMS detector.

At CDF the input objects to jet reconstruction are PhysicsTowers. There are 52 physics towers in η and 24 physics towers in ϕ. The ϕ segmentation of the PhysicsTowers (15^o) correspond to the segmentation of the calorimeter in the central region. In the endplug, the calorimeter segmentation in ϕ is finer (5^o) and the PhysicsTowers correspond to a sum of three calorimeter towers. The energy of a physics tower is always a sum of the energies of the calorimeter towers it contains. The angles are calculated from energy centroids and the event vertex, which can be chosen by its own algorithm. Thus the tower transverse energies and η values are provided from the desired event vertex, allowing one to form jet objects that can be more directly used for physics purposes. For objects other than towers, like MC particles, the inputs are clustered in the relatively fine segmentation of the PhysicsTowers to provide a uniform input for the jet clustering algorithm. The PhysicsTowers are then stored in a two dimensional structure, 52 in η by 24 in ϕ, allowing the most rapid possible lookup of towers (directly via the indices) and their nearest neighbors (by stepping 1 in η or ϕ). This greatly speeds up the jet clustering code for the most important application: reconstruction of jets from the calorimeter towers. The enhanced efficiency is beneficial to both the high level trigger (software trigger) and the effort to re-process large samples with newer versions of the jet clustering code. We believe that CMS will also benefit from a grid-like storage strategy for it's calorimeter towers, although not for other inputs to jet reconstruction.

At D0 the input objects to jet reconstruction are four-vectors, whether from CalTowers or from MC particles. Similar to CDF, the algorithm which creates the CalTowers also includes the value of the event vertex when calculating angles and hence transverse energies and η. However, at D0, since the jet algorithms run directly off four-vectors they take the MC particles directly as an input, and do not construct intermediate towers out of them. At CMS this is the preferred approach for clustering tracks or MC particles into jets. For clustering charged or neutral pion candidates in the calorimeter into jets, clustering the four-vectors of the candidates is also a more natural approach, because the pion candidates will frequently be larger than a single tower in the HCAL. We recommend that CMS preserve it ability to cluster four-vectors.

CDF and D0 both determined early on that it is important to use the event vertex when determining the angle of the energy deposition with respect to the beam, since a non-zero value of the z position of the event vertex will greatly affect the angle, and hence both the transverse energy deposition and the physics η of that deposition. While this is absolutely critical at the Tevatron, where the beam spot has a width of $\sigma = 25$ cm, and an offset of a few cm from the center of the detector, it is still important at the LHC where the beam spot has a width of $\sigma = 5$ cm, and an unknown offset. For example, we will likely analyze many events that come from an event vertex of z=10 cm, and at a radius of ~ 2 meters to not account for the event vertex introduces an error of 5% in the tower angle, and therefore an error of 5% in the transverse energy. We strongly recommend that CMS calorimeter software take

into account the event vertex when calculating transverse energies and physics η of the towers. In short, we should produce a PhysicsTower that is an EcalPlusHcalTower from an adjustable event vertex.

The CMS flexible interface between the jet input and the jet objects, the VJetableObjects, makes it difficult to save with the jet a set of links back to the input objects. For rapidly constructing jets from PhysicsTowers stored in a two-dimensional structure, it doesn't make sense to have intermediate VJetableObjects. Neither CDF or D0 found it necessary to introduce a flexible interface, a VJetableObject, between the jet and its input. We don't believe that when we start taking data CMS will make constructive use of all the jet algorithms it currently employs for which a flexible interface was necessary. However, for constructing jets from four-vectors the VJetableObjects are a reasonable interim solution. VJetable objects should be maintained until it becomes necessary to avoid the potential duplication of storage of the four-vectors, and until we decide which algorithm we must in the end support. We recommend supporting VJetableObjects as a CMS legacy interface for existing algorithms, and for further experimentation with algorithms based on tracking and pre-clustering of charged and neutral pion candidates in the calorimeter. We recommend that RecJets contain direct links back to the PhysicsTowers for the default CMS algorithm. This recommendation will not result in the most elegant code, but it is intended to produce efficient jet clustering code from calorimeter towers, while supporting ongoing jet clustering based on four-vectors.

4 Jet Correction

Reconstructed jets require energy corrections as a function of P_T and η in order to be used for analysis purposes. At CMS it has been proposed that these corrections be applied before the HLT trigger, potentially making them even more critical. We have concentrated in this review on the jet reconstruction and access methods, because they will lay the foundation for jets, giving us something that is understood and worth correcting. Nevertheless, we believe that jet corrections are important, and they should be part of the official CMS software. Since the jet corrections are intimately part of the detailed analysis that uses jets, they often reside outside of the jet package itself, depending on many different reconstruction packages for their input. The corrections are also developed by users, and evolve quickly with the time dependent understanding of the CMS detector, in contrast to the reconstruction software itself which must be as stable as possible. The CMS jet software has a mechanism for jet corrections, called calibrations, which started inside the Jets package and is in the process of being moved outside the package. The CDF and D0 code similarly have jet corrections that are located outside of the reconstruction package itself. After the jet reconstruction code is stable, and the default algorithm that CMS will use for jets has been developed, CMS should review the jet correction code and we are willing to help.

5 Conclusions

We have reviewed the jet code at CMS. We have the following recommendations

1. Towers: a new tower object, PhysicsTower, should be developed that is an EcalPlusHcalTower from an event vertex that is user adjustable, and should be stored in a two dimensional eta-phi array for the rapidest possible reconstruction code that is aware of the tower geometry.
2. Algorithm: a cone algorithm that includes splitting and merging (we recommend the CDF/D0 midpoint algorithm) should be implemented using PhysicsTowers and should be adopted as the CMS default jet algorithm.
3. Variables: expand the list of quantities accessible through the stored jet to include the full list in Appendix A.
4. Constituents: the jet constituents should be replaced with a list of pointers to the jet input objects, and methods that use these pointers, to allow full tracing back to the lowest level of persistent data without duplicate storage.
5. Four-vectors: jets based on clustering arbitrary four-vectors, not tied to a tower geometry, should continue to be supported by an interface such as the current VJetableObject.

We plan to build a prototype version of the jet code that incorporates as many of these recommendations as possible. We welcome comments from the collaboration.

A Jet Variables

The text below defines the variables that should be pre-calculated and stored with the jet, and the variables that should be available through the jet class but not stored with the jet. We propose nine long words to store with each jet (Px, Py, Pz, E, Ntower, Ntrack, EM Fraction, Charged Fraction and Detector Pseudorapidity) and nine other variables to calculate from the stored quantities or from links back to the jet constituents (P, Pt, Et, M, Phi, Pseudorapidity, Rapidity, Max EM Tower, Max Had Tower).

```
1) The Jet four-vector as a true Lorentz vector: Px, Py, Pz and E.
   These are calculated and stored with the jet.
   a) Px: the sum of Px_i over jet constituents (i=towers, tracks,
      or MC particles). For towers Px_i = E_i Sin Theta_i Cos Phi_i
   b) Py: the sum of Py_i over jet constituents
      For towers Py_i = E_i Sin Theta_i Sin Phi_i
   c) Pz: the sum of Pz_i over jet constituents
      For towers Pz = E_i Cos Theta_i
   d) E: the sum of E_i over jet constituents

2) Standard quantities derived from the four-vectors, but
   returned by the jet for the sake of uniformity of use.
   These are not stored with the jet, but are calculated by
   methods of the jet class from the stored four-vector in 1).
   a) P = sqrt (Px**2 + Py**2 + Pz**2)
   b) Pt = sqrt (Px**2 + Py**2)
   c) Et = E * Pt/P
   d) M = E**2 - P**2
   e) Phi = ArcTangent (Py/Px) (defined to go from 0 to 2pi in radians)
   f) Pseudorapidity Eta = 0.5 * Log[(P +Pz)/(P-Pz)]
   g) Rapidity = 0.5 * Log[(E + Pz)/(E - Pz)]

3) Diagnostic Jet Properties.  The first two are calculated by a method
   that goes back to the input towers, because these variables
   will only be used occasionally, and the rest are stored with the jet
   because they will be used more frequently.
   a) Max EM Tower: maximum energy in an EM tower in the jet.
   b) Max HAD Tower: maximum energy in a HAD tower in the jet.
   c) Ntower: Number of towers (or constituents) that compose the jet.
   d) Ntrack: Number of tracks pointing at the towers of the jet, in the
              case where the jet is made of towers.
   e) Charged Fraction: vector sum of PT of tracks that form Ntrack divided
                        by the Pt of the jet.
   f) EM Fraction: sum of energy of EM towers in the jet divided
                   by the total jet Energy.

4) Detector Pseudorapidity.  This is stored with the jet.
   Pseudorapidity as above, but calculated with
   respect to the center of the detector: assuming the Z position of the
   event vertex = 0 exactly.  The eta cracks of the detector most naturally
   show up in this variable.

   Except for Detector Pseudorapidity, the quantities in 1) and 2) are
   calculated with respect to the assumed Z position of the event vertex.
   This is something that we have to be able to change, running the jet
   finding with different assumed values of the Z component of the event
   vertex:  some common assumptions are
          Z = 0 before tracking is working.
          Z = Z position of the the vertex with the most number of tracks
              pointing at it.
```

```
        Z = Z position of the vertex with the highest Pt in tracks
            pointing at it.
        Z = Z position of the vertex with the tracks from the given jet
            pointing at it.
        etc.
```

The discussion above should alert us all to a subtlety about the definitions of Theta_i and Phi_i above. These are the angles of the energy deposition in the towers (or jet constituents) with respect to the z coordinate of the event vertex, and when the event vertex is not at the center of the detector they should in principle be defined with respect to some depth within the calorimeter. Eventually, I would calculate the angles separately for each of the EM and HADRONIC towers, making the depth some average position for each of the showers. One can get even fancier and have the depth evolve with energy. The main thing is to make sure that we have the ability to separately calculate a pseudorapidity with respect to the event vertex (2f above) and a pseudorapidity with respect to the center of the detector (4), and realize they are potentially different. The pseudorapidity with respect to the event vertex (2f) is needed for physics studies, while the pseudorapidity with respect to the center of the detector (4) is needed for studies of the response of the detector in different regions, understanding and correcting for cracks and miscalibrations, etc.

References

[1] **CMS Note 1999/034**, J.P. Wellisch, *An Object Oriented Jet Finder Library*, 19 May, 1999.

[2] Arno Heister, CMS jet code expert, private communications.

[3] Pierre Savard, CDF jet code expert, private communication.

[4] Emmanuel Busato, D0 jet code expert, private communication.